View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Washington University St. Louis: Open Scholarship

Washington University in St. Louis

Washington University Open Scholarship

All Computer Science and Engineering Research

Computer Science and Engineering

Report Number: WUCS-01-30

2001-01-01

Indra: A Distributed Approach to Network Intrusion Detection and Prevention

Qi Zhang and Ramaprabhu Janakiraman

While advances in computer and communications technology have made the network ubiquitous, they ahve also rendered networked systems vulnerable to malicious attacks orchestrated from a distance. These attacks, usually called cracker attacks or intrusions, start with crackers infiltrating a network through a vulnerable host and then going on to launch further attacks. Crackers depend on increasingly sophisticated techniques like using distributed attack sources. On the other hand, software that guards against them remains rooted in traditional centralized techniques, presenting an easily-targetable single point of failure. Scalable, distributed network intrusion prevention software is sorely needed. We propose Indra - a... Read complete abstract on page 2.

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

 Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation

Zhang, Qi and Janakiraman, Ramaprabhu, "Indra: A Distributed Approach to Network Intrusion Detection and Prevention" Report Number: WUCS-01-30 (2001). *All Computer Science and Engineering Research.*
https://openscholarship.wustl.edu/cse_research/268

Department of Computer Science & Engineering - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160.

This technical report is available at Washington University Open Scholarship: https://openscholarship.wustl.edu/cse_research/268

Indra: A Distributed Approach to Network Intrusion Detection and Prevention

Qi Zhang and Ramaprabhu Janakiraman

Complete Abstract:

While advances in computer and communications technology have made the network ubiquitous, they ahve also rendered networked systems vulnerable to malicious attacks orchestrated from a distance. These attacks, usually called cracker attacks or intrusions, start with crackers infiltrating a network through a vulnerable host and then going on to launch further attacks. Crackers depend on increasingly sophisticated techniques like using distributed attack sources. On the other hand, software that guards against them remains rooted in traditional centralized techniques, presenting an easily-targetable single point of failure. Scalable, distributed network intrusion prevention software is sorely needed. We propose Indra - a distributed scheme that depends on sharing information between trusted peers in a network to guard the network as a whole against intrusion attempts. We futher describe a plugin mechanism that enables an administrator to simultaneously plug weaknesses in thousands of machines with a single E-Mail.

Indra: A Distributed Approach to Network Intrusion Detection and Prevention

Qi Zhang and Ramaprabhu Janakiraman

WUCS-01-30

October 2001

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130

Indra: A Distributed Approach to Network Intrusion Detection and Prevention

Qi Zhang
Department of Computer Science
qz@cs.wustl.edu

Ramaprabhu Janakiraman
Applied Research Laboratory
rama@arl.wustl.edu

Washington University in St. Louis
Missouri - 63130-4899, USA

Washington University Technical Report # WUCS-01-30

Abstract—**While advances in computer and communications technology have made the network ubiquitous, they have also rendered networked systems vulnerable to malicious attacks orchestrated from a distance. These attacks, usually called cracker attacks or intrusions, start with crackers infiltrating a network through a vulnerable host and then going on to launch further attacks. Crackers depend on increasingly sophisticated techniques like using distributed attack sources. On the other hand, software that guards against them remains rooted in traditional centralized techniques, presenting an easily-targetable single point of failure. Scalable, distributed network intrusion prevention software is sorely needed.**

We propose Indra – a distributed scheme that depends on sharing information between trusted peers in a network to guard the network as a whole against intrusion attempts. We further describe a plugin mechanism that enables an administrator to simultaneously plug weaknesses in thousands of machines with a single E-Mail.

I. Introduction

A. Background

Intrusion is the act or attempted act of using a computer system or computer resources without the requisite privileges, causing willful or incidental damage. *Intrusion detection* involves identifying individuals or machines that perform or attempt intrusion. *Intrusion Detection Systems* (IDS) are computer programs that attempt to perform intrusion detection by studying the behavior of intruding processes, preferably in real-time. Intrusion is primarily a network based activity. With increasing global network connectivity, the topic of intrusion has gained prominence, spurring active research on efficient IDS.

Intrusion detection systems can be classified on the basis of a multitude of factors. Some significant ones, described in greater detail in [1] are:

Response to Intrusion This can be passive or active. A passive system is content with just detecting intrusion, leaving its handling to a human agency. On the other hand, an active system takes action, for example terminating network connections to a suspected host. Obviously, active systems are much better scalable and responsive, but open themselves up to denial of service attacks by overreacting to deliberately triggered false alarms.

Source of audit data The data to be examined can be network data (network packets etc.) or host data (application logs, system call traces etc.)

Locus of data collection and processing Data-collection can be centralized or distributed. Again, this data can be processed centrally or at distributed locations.

In recent times, there has been a lot of interest in distributed schemes for Intrusion detection. While the research community has been active in this area [2–8], most existing schemes are passive in the sense that they only implement collecting information in a distributed manner. The controlling intelligence is centralized in the person of the system administrator managing the administrative domain.

Indra, on the other hand, can be configured to automatically make access control decisions, based on dynamically generated information shared between trusted peers. Indra also has a unique plugin mechanism by which the administrator can securely multicast plugins to multiple hosts, which are then loaded on the fly. These features lead to a high degree of scalability, both in terms of number of hosts and number of exploits to guard against. This seems particularly desirable, since new exploits are unearthed virtually on a daily basis.

B. Design Goals

Project Indra is named after an Indian God credited with a protective function. It stands for INtrusion Detection and Rapid Action. The phrase describes its goal and functionality with surprising accuracy, given the fact that the acronym was retro-fitted, as is frequently the case with project names that have a nice ring to them.

Indra is an intrusion detection tool that takes a proactive and distributed approach to network security. It is often the case that crackers try out common exploits on different machines, hoping to stumble upon a machine on which a particular vulnerability is extant. Sometimes these attacks are detected and repulsed by intrusion detection software in place on a particular machine. But a persistent attacker eventually manages to find a weak link in the chain.

Earlier research [9] seems to indicate that it usually takes a large number (as many as 1000) of crack attempts for an attack to be successful. The broad goals of project Indra is to distribute such attempt information gathered by the intended victim(s), among all nodes in a "trusted" network (this term will be elaborated on shortly). The intention is to put together an

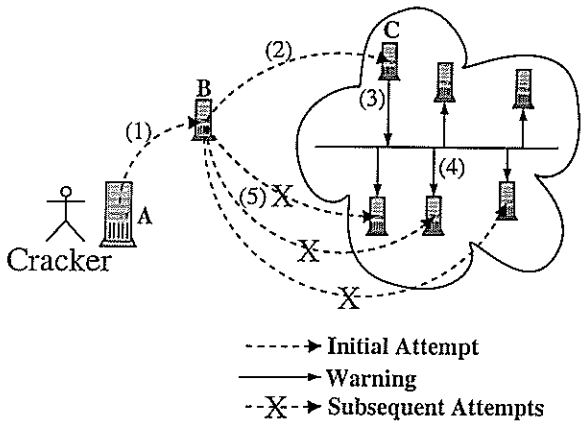

Fig. 1. Overview of Indra

intrusion detection system with these properties:

Autonomous: Minimal need for human monitoring

Distributed: No need for central controller

Extensible: New Security policies can be plugged in dynamically

Flexible: Should support a wide range range of configuration options based on varying levels of security

Secure: Should use demonstrably secure cryptographic and authentication primitives

Learning: The network as a whole should become wiser when one of its hosts learns about an attempted intrusion.

C. Outline of this paper

We start by sketching the conceptual overview of this paper. We then describe the architecture of the Indra system. We propose a plugin mechanism that provides for dynamic extensibility in Indra. We describe its key distribution mechanism. We discuss the relation of our work with other research in this area. We conclude with an outline of future improvements.

II. Conceptual Overview

At its core, the design of Indra can be described by Figure 1: Each host on the trusted network runs a special security daemon called Indra daemons, which both watches out for intrusion attempts and also enforces access control based on its memory of earlier attempts. By trusted network, we mean: each host on the network can be authenticated by cryptographic means such as public-key cryptography by other hosts on the network. In particular, as shown in Figure 1, the following sequence of events occur:

1. The cracker on A finds the weak access point B in the network.
2. The cracker initiates attacks from B [1] to hosts in the trusted network to which the host C is connected. It is assumed that all hosts in the network, including C, run Indra daemons.
3. The Indra daemon at C detects the attack from B and then multicast a secure warning message regarding B to all its trusted neighors.
4. Each Indra daemon receives the message from C, verifies its integrity and then places B on a 'black-list' of suspected intrusion sources.
5. The cracker, having failed in his attempt on C, tries it out with other hosts in the same domain. These subsequent attacks are repelled straightaway by the forewarned hosts.

While this situation is ideal and easy to spell out, it presents practical difficulties at various levels that have to be overcome:

Transport: How do the daemons communicate with each other? How do they transmit a message to all the other daemons? Some communication model has to be devised.

Trust: How do the daemons trust messages and their senders? Obviously, messages have varying importance depending on who sends them.

Policy: Suppose intrusion is suspected. How do the daemons react to it? Solutions can range from paranoia to indifference.

In the next few sections, we deal with each of these in turn.

III. Architecture of Indra

A. Design Philosophy

Indra was designed to evolve gracefully; It consists of a collection of reusable components that interact through well defined interfaces. Some of the driving principles behind Indra's design are:

Separation of Policy and Mechanism We have tried to implement, as independent entities, a cryptographic mechanism that provides support for asynchronous messaging and a security policy that uses this mechanism as a primitive. The cryptographic layer is useful in its own right. For example, it could be used to implement with relative ease, a secure group communication application or a peer-to-peer file-sharing utility.

Extensibility One particularly novel feature of Indra is that it allows pluggable security policies. Like Antivirus software, intrusion detection software undergoes continuous incremental change. New, imaginative, exploits are devised all the time. Indra handles this by allowing the system administrator to :

- write, with very little effort, new Java modules that can be plugged into our security daemons.
- securely transmit the compiled Java class files to all security daemons on the network.

Flexibility Indra allows for fine grained control and configuration. Security concerns differ widely and it is important that administrators are not stuck with an all-or-nothing security policy. We have also made it very easy to replace our cryptographic modules with any other standard ones, as easy as changing a couple of values in a property file.

As can be seen from Figure 2, the architecture of Indra consists of these three distinct layers:

Transport Layer: This provides transport primitives like unicast and multicast messaging to the upper layers.

Crypto Layer: This implements secure messaging through the use of strong cryptography.

Daemon Layer: This layer, which exists as a collection of daemons, implements the core functionality of Indra.

[1] or a sequence of such B's

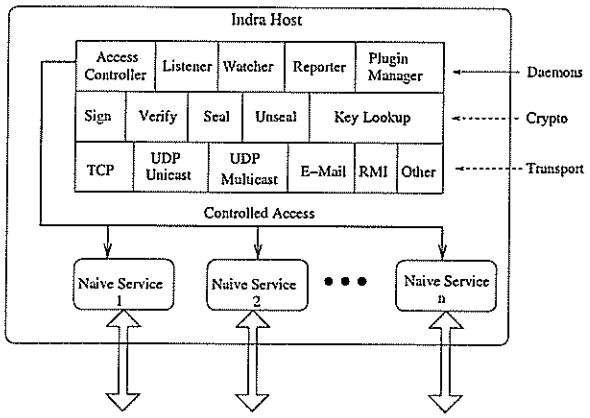

Fig. 2. Indra Architecture

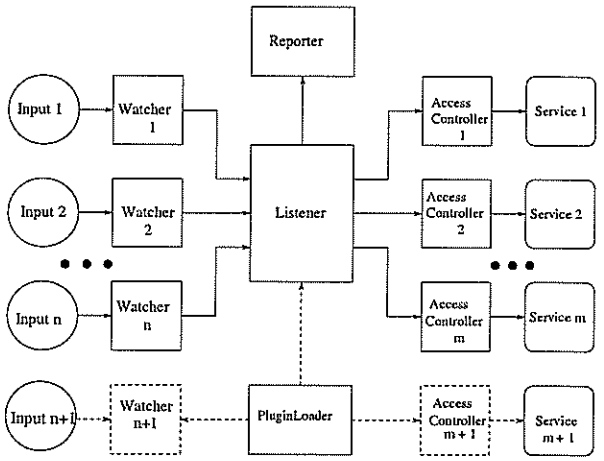

Fig. 3. Indra Daemons

B. *Transport Primitives*

At the lowest layer, Indra uses a bunch of transport primitives for communication. Peer-to-Peer communication is through TCP or UDP unicast. Group communication, e.g. for warning messages, is through IP Multicast. For situations in which human interaction is necessary, like when new plugins are to be deployed, an E-Mail gateway is provided. This means the administrator can communicate securely through E-Mail with all the Indra daemons on the trusted network.

C. *A Secure Asynchronous Messaging API*

In an IDS, it would be especially imprudent to act upon unverifiable information originating from untrusted sources. It is advisable that the only trusted hosts are those that belong to the same administrative domain and whose identity can be verified using a standard cryptographic signature scheme. Thus, in order to prevent intruders from exploiting the system by faking messages, both sender authenticity and message integrity should be assured.

We have designed and implemented a comprehensive library that provides support for secure [2], asynchronous group messaging. The delivery model is based on a callback mechanism, by which potential message receivers register themselves with message agents. When messages addressed to a registered recipient arrive, the message agent asynchronously delivers the message to the recipient, optionally verifying signed messages and decrypting encrypted content.

Implementation-wise, we use message digests (MD5) and digital signatures (MD5/RSA with 1024-bit keys) to ensure the integrity and authenticity of all messages. The sender uses MD5 to digest the message and signs it with its private key and the receiver verifies the message with the sender's public key. We use the Java Cryptography Architecture and extension for this part. [10] The library makes it trivial to implement messaging applications that require secure data delivery. The Indra daemons depend on this for their functioning.

D. *Indra Daemons*

At the topmost level, all the functionality of Indra is achieved by a set of daemons which, in our implementation, correspond to Java threads. These daemons belong to one of the following classes.

Watchers: These are the first level daemons that are on the outlook for any suspicious activity, either on the local system or over the network, for example multiple failed login attempts, port-scan attempts or suspicious system-call sequences.

Access Controllers: These daemons provided controlled access to resources. The control is dynamic and depends on what the listeners tell them to do. When they get a warning against a particular user-id on a machine, they selectively filter out access to that particular (account, machine) combination. For determining accounts, it uses RFC 1413 [11], or the ident protocol. This is not done (since there is no point anyway) if the attacker originates outside the trusted network.

Listeners: These are daemons that listen to the watchers. Listeners aggregate the warnings that are generated by the Watchers. Then based on the security level or any other policy dictated by the admin, the listeners convey the warnings to the Access Controllers. Listeners are essentially selective filters that stand between the watchers and access controllers.

If watchers were sense organs and access controllers limbs, the listener would be the central intelligence that drives motor function based on sensory input. For example, certain kinds of exploit attempts might result in vulnerable services being denied while other, presumably secure, services continue to operate normally.

Reporters: These daemons are responsible for communicating with other hosts, either receiving warnings and passing them on to the listeners or aggregating warnings from listeners and passing them along the network to other hosts.

The daemons could be configured by the system administrator for different levels of security. For example, a host with critical information could be configured to deny all network connections to a machine which is identified as an originator of repeated failed logins. At another level, routers could run security

[2] Encryption and authentication

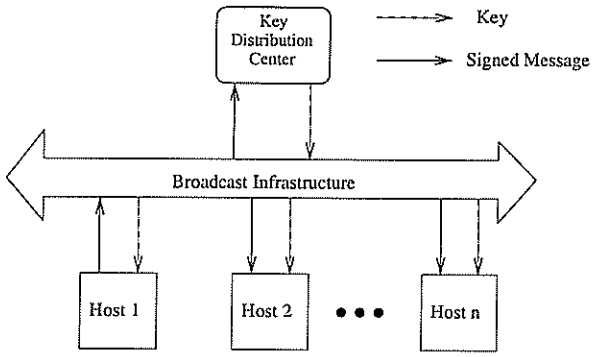

Fig. 4. Key Distribution in Indra

agents that cut off packets that originate from a compromised machine, effectively isolating the machine from the network. Instead of taking it upon itself to make all these decisions, Indra provides a scaffold or framework that allows these options to be implemented by the administrator with ease.

E. The Plugin Mechanism

Indra provides a mechanism by which additional daemons [3] can be plugged in at run-time into the Indra system. Whenever the administrator needs to change the security policy, either because a new exploit has surfaced or the security concerns have changed, she can write Java code that implements the necessary functionality and E-Mail or multicast it to one or more [4] Indra daemons. These modules will be authenticated against the administrator's public key by the Plugin manager and then dynamically loaded into the daemon's address space.

We find that using Java for our implementation serves us well here. Code that compiles to native machine code, with its ability to forge pointers to arbitrary memory locations and to execute any combination of native machine instructions, is extremely difficult to audit or validate. Java, with its concept of a virtual machine as a sandbox, allows fine grained access control to resources, enabling different security policies for inbuilt code and code that is loaded over the network. This is analogous to executing Java applets securely inside the context of a browser.

IV. Key Distribution in Indra

Indra depends on public key servers when using public key cryptography. To simply the design and reduce the load on the endpoints running Indra, a centralized, trusted Key Distribution Center (KDC) is used in our design. Since Indra hosts communicate through multicast, potentially every host on the trusted network will get messages addressed to the Indra group, which it will have to subsequently validate against the sender's public key. This imposes heavy and bursty load on the KDC.

In Indra, this is solved like so: The KDC listens in to the Indra traffic like any other host on the network. When it receives a signed message, it preemptively multicasts the sender's public key[5], saving network and KDC overhead and speeding up the intrusion detection process. This is illustrated in Figure 4.

[3] Watchers, Listeners or AccessControllers
[4] possibly all
[5] of course signing it with its private key.

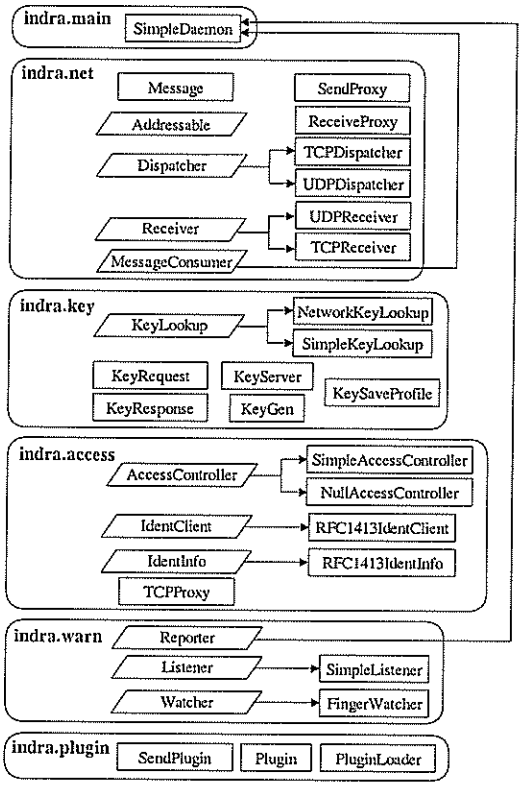

Fig. 5. Java Class hierarchy for Indra

V. Implementation Details

Figure 5 shows the class hierarchy and dependencies among the various packages in Indra. The master daemon in Indra is imaginatively called *SimpleDaemon*. Its role is to startup the various Watchers, Listeners, AccessControllers from a startup file. Of course, this can be configured on the fly. After the startup phase, it takes to acting as the Reporter, serving as the single point of contact for peer Indra daemons.

There is also a standalone Keyserver or Key Distribution Center, which can be set up to serve public keys from a presetup database of keys over TCP and UDP.

VI. Limitations and Work in Progress

Indra is very much work-in-progress. We have a prototype implementation working, but it is currently not too strong in the intrusion detection department. For example, we use simple port-logging or failed-login counts as indicators of intrusion attempts. Overall, the fundamental contribution of Indra is not that of new intrusion detection techniques. Instead, we have tried to provide a framework that complements these techniques and help them maintain relevance in a massively networked world.

Our prototype implementation of Indra is Java. While Java provides Indra system adaptability and a mechanism for automatic update of plugins, it also suffers a performance hit compared to the system programming language of choice.

We also believe that for production quality, we need to step

down a few notches from the pure application-layer stand we have taken. For example, efficient repulsion of Distributed Denial of Service (DDOS) attacks might require hooks into the network layer.

Indra also depends on the host operating system for certain primitives like process address space and sanctity of 'root' user. We plan to explore ways in which Indra might interoperate with operating system kernels. This might need security enhancements to current OS kernels.

For simplicity, we have assumed the presence of a trusted centralized keyserver. Extending our discussion to work with a distributed keyserver network is straightforward. Furthermore, we have used our own keyserver implementation with our own simple protocol. Interoperability with existing keyserver protocols like PGP is a must.

VII. Related work

The idea of using distributed intrusion detection has been proposed with several variations over the past decade. Schemes have been proposed using distributed data collection and, in relatively fewer cases, distributed analysis agents.

An interesting approach to this problem using concepts of Immunology is [12]

The Distributed Firewall scheme [13] proposes a central access control access policy which is enforced by individual endpoints.

The NADIR system [2] uses distributed data collection and centralized analysis by an expert system.

The GrIDS project [3] is a 'hierarchial graph-based IDS capable of doing distributed data collection and analysis in large networks.' GrIDS uses data source modules running in each host to extract information, which is used by graph engines to build a graph representation of network activity. GrIDS is used to detect large-scale automated attacks on networked systems. GrIDS is again a purely a passive detection-based scheme, corrective action presumably left to the system administrator.

AAFID architecture [4] describes a distributed IDS based on which is based on multiple autonomous agents. These agents can be added and removed from a system on the fly, without affecting other components or bringing down the whole IDS. There is no facility for automated handling of Intrusions, i.e AAFID is a passive IDS.

The two schemes that are most closely related to Indra are Cooperating Security Managers (CSM) [5] and EMERALD [6].

CSM is an peer based IDS designed for use in a distributed network environment. Each CSM acts like a host-based local IDS for its host, while additionally cooperating with other CSMs without the use of a central controller.

EMERALD is a powerful distributed IDS that is active and distributed. However, it does not seem to support on-the-fly plugin upgrades.

Host based IDS's are offered by a lot of software packages. Tripwire , Satan and Titan [14, 15] are notable examples. They are complementary to Indra and could be easily integrated with it as Watcher daemons.

VIII. Summary

As the global Internet becomes increasingly pervasive, Computer Intrusion and its prevention assumes greater importance. As network sizes explode, it is imperative that Intrusion Detection systems be distributed and self-maintaining in order to maintain scalability. In this paper, we describe the design and implementation of such a scheme, Indra, which promises to scale well under increasing network sizes and more determined attackers.

At the frenetic pace at which software is written and deployed over the network, new vulnerabilities in networked systems crop up as fast as older ones are detected and plugged. In such a scenario, protection systems need to be pluggable to have any chance of keeping up with the latest bug-reports. Indra offers a scalable solution by providing for security plugins that can be loaded on the fly simulataneusly by thousands of machines in an administrative domain.

Further, we are working on a standard and flexible interface to writing security plugins for Indra. Ultimately, this would enable an advisory agency [16] to write plugin modules as soon as a vulnerabilility is detected, sign and dispatch them to a hierarchy of mailing-lists in which subscribed *machines* are members. Alternately, an efficient multicast transport mechanism like SRM [17] or ALMI [18] could be used, if and when such mechanisms are widely deployed over the Internet. In any case, we predict turn-around time to be of the order of a few minutes, for machines distributed throughout the Internet.

References

[1] S. Axelsson. Research in intrusion-detection systems: A survey. Technical Report 98–17, Dept. of Computer Eng. Chalmers Univ. of Tech, SE-412 96, Goteborg, Sweden, December 1998.

[2] et al. Hochberg. Nadir: An automated system for detecting network intrusion and misuse. In *Computers & Security. 1993. Elsevier Science, New York*, pages 235–248, 1993.

[3] S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, J. Rowe, S. Staniford-Chen, R. Yip, and D. Zerkle. The design of grids: A graph-based intrusion detection system. Technical Report CSE-99-2, U.C. Davis Computer Science Department, January 1999.

[4] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, E. Spafford, and D.Zamboni. An architecture for intrusion detection using autonomous agents. Technical Report 98/05, Purdue University, 1998.

[5] G. White, E. Fisch, and U. Pooch. Cooperating security managers: A peer-based intrusion detection system. In *IEEE Network*, volume 10(1), pages 20–23, Jan./Feb. 1994.

[6] P. A. Porras and P. G. Neumann. Emerald: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the 20th National Information Systems Security Conference*, pages 353–365, October 1997.

[7] G. Helmer, J. Wong, V. Honavar, and L. Miller. Intelligent agents for intrusion detection. In *Proceedings, IEEE Information Technology Conference*, pages 121–124, September 1998.

[8] M. Crosbie and G. Spafford. Defending a computer system using autonomous agents. Technical Report 95-022, Dept. of Computer Sciences, Purdue University, Mar 1996.

[9] J. Howard. *An Analysis of Security Incidents on the Internet.* PhD thesis, Carnegie Mellon University, 1998.

[10] Java cryptography. http://www.javasoft.com/products/jce/.

[11] Rfc 1413 – the ident daemon. http://www.faqs.org/rfcs/rfc1413.html.

[12] S. Forrest, S. Hofmeyr, and A. Somayaji. Computer immunology. In *Communications of the ACM, (submitted Dec. 1996)*, 1996.

[13] S. Ioannidis, A. Keromytis, S. Bellovin, and J. Smith. Implementing a distributed firewall. In *Proceedings of Computer and Communications Security (CCS)*, November 2000.

[14] Satan. http://www.fish.com/satan/.

[15] Titan. http://www.fish.com/titan/.

[16] Cert coordination center. http://www.cert.org.

[17] Sneha Kumar Kasera, James F. Kurose, and Donald F. Towsley. Scalable reliable multicast using multiple multicast groups. In *Measurement and Modeling of Computer Systems*, pages 64–74, 1997.

[18] Dimitris Pendarakis, Sherlia Shi, Dinesh Verma, and Marcel Waldvogel. ALMI: An application level multicast infrastructure. In *3rd USNIX Symposium on Internet Technologies and Systems (USITS '01)*, San Francisco, CA, USA, March 2001.